УДК 004

Работа с многопоточностью в Java

Семченко Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассмотрена работа с многопоточность в Java. Написаны примерные скрипты описывающие работу многопоточности. Конечным результатом является выполненный обзор работы с многопоточностью на языке программирования Java.
Ключевые слова: Многопоточность, Java, ядра

Working with multithreading in Java

Semchenko Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article covers working with multithreading in Java. Sample scripts have been written to describe how multithreading works. The end result is a completed overview of working with multithreading in the Java programming language.
Keywords: Multithreading, Java, core

Поток - это легковесный процесс. В процессе может быть запущено множество потоков. При одновременном выполнении разных частей программы в разных потоках поможет улучшить скорость отклика системы.

Цель статьи выполнить обзор на работу многопоточности на языке программирования Java.

А.В.Шлапкин, Т.Г.Султанов рассмотрели в своей работе вопрос о применении многопоточности в однопоточные программы на языке программирования Java [1]. З.З. Гасанов рассмотрел подход к программированию многопоточных программ на языках программирования Java и Go [2]. Д.А.Кравченко разработал многопользовательский клиент-

серверный чат с использованием технологии многопоточности [3]. В.В.Гринеев, Д.А.Кушаль, В.А.Шарапович разработали простой в использовании программный пакет на базе Java CelNetAnalyzer [4].

Многопоточность позволяет запускать несколько потоков одновременно. Например, в веб-браузере может быть поток, который обрабатывает пользовательский интерфейс, и параллельно может быть еще один поток, который выбирает данные для отображения. Таким образом, многопоточность улучшает отзывчивость системы.

В одноядерной системе имеется планировщик потоков, предоставляемый JVM, который решает, какой поток запускать в любой момент времени. Планировщик дает каждому потоку небольшой отрезок времени. Таким образом, в любой момент времени есть только один поток, который фактически выполняется в процессоре. Но из-за разделения времени возникает ощущение, что несколько потоков работают одновременно.

Даже в многоядерных системах задействован планировщик потоков. Но поскольку в системе есть несколько ядер, то можно иметь несколько потоков, работающих в одно и то же время. Например, если есть двуядерная система, то может быть 2 потока, работающих в одно и то же время. Первый поток будет выполняться в первом ядре, а второй поток будет выполняться во втором ядре.

Многопоточность позволяет улучшить отзывчивость системы. Например, в веб-браузере, если все работает в одном потоке, система будет не отвечать всякий раз, когда данные выбираются для отображения. Если для получения данных требуется 10 секунд, то за эти 10 секунд не будет возможности делать что-либо еще в браузере, например, открывать новые вкладки или даже закрывать веб-браузер.

Потоки в java создаются, используя следующие задачи: расширение класса потока, реализация исполняемого интерфейса, реализация вызываемого интерфейса.

Для написания кода, который можно запускать в потоке, создадим класс, а затем расширяем класс «Thread». Задача, выполняемая этим фрагментом кода, должна быть помещена в функцию «run()». В приведенном ниже коде можно видеть, что «Worker» - это класс, расширяющий класс «Thread», а задача печати чисел от 0 до 5 выполняется внутри функции «run()». Метод «Thread.currentThread().GetName()» используется для получения имени текущего потока, в котором выполняется код (рис.1).

```
class Worker extends Thread {

    @Override
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName()
                    + ": " + i);
        }
    }

}
```

Рисунок 1 – Метод «Thread»

Чтобы создать поток, нужно создать экземпляр класса «Worker». Создадим три потока «t1», «t2» и «t3» из класса «Worker», затем запустим потоки с помощью функции «start()» (рис.2).

```
public class ThreadClassDemo {
    public static void main(String[] args) {
        Thread t1 = new Worker();
        Thread t2 = new Worker();
        Thread t3 = new Worker();
        t1.start();
        t2.start();
        t3.start();

    }
}
```

Рисунок 2 – Создание потоков

Вот результат, который получится, выполнив приведенный выше код (рис.3).

```
<terminated> ThreadClassDemo
Thread-1: 0
Thread-2: 0
Thread-0: 0
Thread-2: 1
Thread-1: 1
Thread-2: 2
Thread-0: 1
Thread-2: 3
Thread-1: 2
Thread-2: 4
Thread-0: 2
Thread-2: 5
Thread-1: 3
Thread-1: 4
Thread-1: 5
Thread-0: 3
Thread-0: 4
Thread-0: 5
```

Рисунок 3 – Вывод в консоль

Видно, что все три потока напечатали числа от 0 до 5. Из вывода видно, чтотри потока не выполняются в какой-либо конкретной последовательности.

Теперь попробуем реализовать многопоточность через интерфейс «Runnable».

Чтобы создать код, который может быть запущен в потоке, создадим класс, а затем реализуем интерфейс «Runnable». Задача, выполняемая этим фрагментом кода, должна быть помещена в функцию «run()».Чтобы создать поток, сначала нужно создать экземпляр «RunnableWorker», который реализует интерфейс «Runnable», а задача печати чисел от 0 до 4 выполняется внутри функции «run ()».

Затем можно создать новый поток, создав экземпляр класса «Thread» и передав экземпляр «RunnableWorker» в качестве аргумента (рис.4).

```
class RunnableWorker implements Runnable{

    @Override
    public void run() {
        for (int i = 0; i <= 4; i++) {
            System.out.println(Thread.currentThread().getName() + ": " + i);
        }
    }

}

public class RunnableInterfaceDemo {

    public static void main(String[] args) {
        Runnable r = new RunnableWorker();
        Thread t1 = new Thread(r);
        Thread t2 = new Thread(r);
        Thread t3 = new Thread(r);

        t1.start();
        t2.start();
        t3.start();

    }

}
```

Рисунок 4 – Метод «Runnable Interface»

Запустив код, получится следующий результат. Последовательность вывода будет меняться каждый раз при запуске кода (рис.5).

```
<terminated> RunnableFunctiona
Thread-2: 0
Thread-0: 0
Thread-1: 0
Thread-0: 1
Thread-2: 1
Thread-0: 2
Thread-1: 1
Thread-0: 3
Thread-2: 2
Thread-0: 4
Thread-1: 2
Thread-2: 3
Thread-1: 3
Thread-2: 4
Thread-1: 4
```

Рисунок 5 – Вывод в консоль

Реализация «Runnable Interface» - лучший вариант, чем расширение класса «Thread», поскольку можно расширить только один класс, а интерфейсов можно реализовывать несколько.

В данной статье была рассмотрена работа с многопоточностью на языке программирования Java. Приведены примеры работ с многопоточностью.

Библиографический список

1. Шлапкин А.В., Султанов Т.Г. Поведение потоков в среде исполнения java // Современные научные исследования и инновации. 2018. № 3-2 (11). С.24-28
2. Гасанов З.З. Анализ производительности многопоточных программ, написанных на языках java и go // Программные продукты и системы. 2019. №2 (114). С. 185-194
3. Кравченко Д.А. Разработка приложения многопользовательского чата на java // Новые информационные технологии в автоматизированных системах. 2018. №17. С. 508-513
4. Гринеев В.В., Кушаль Д.А., Шарапович В.А. CELNETANALYZER: Высокопроизводительный пакет java для топологического анализа сотовых сетей // Новые информационные технологии в автоматизированных системах. 2019. №18. С. 479-481